METHOD AND APPARATUS FOR TRUST-BASED, FINE-GRAINED RATE LIMITING OF NETWORK REQUESTS

CROSS REFERENCE TO RELATED APPLICATION

This application claims benefit of United States Provisional Patent Application Serial Number 60/523,427, filed November 18, 2003.

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

The invention relates generally to network security. More particularly, the invention relates to a method and system for trust-based, fine-grained rate limiting of network requests.

DESCRIPTION OF RELATED TECHNOLOGY

A fundamental feature of current network technologies is the provision of network services to individual users by furnishing each user an account. Users then request services through their accounts by logging into a server using a client. In current usage, the term 'client' may refer either to the software application by which a user accesses the network resources provided from the server, or the machine that is running the client software. To guarantee that the party attempting to gain access to an account is actually the rightful accountholder, access is controlled through an

authentication process, in which the account holder is required to provide one or more pieces of information, known only to the account holder, before he is granted access to network services. Typically, authentication requires provision of a user ID and a password. User accounts are seen in most network environments, *e.g.* local area networks limited to a small company, as well as in ISP's (Internet service provider) having tens of millions of subscribers. Unfortunately, individuals wishing to make unauthorized use of another's user account are common, and thus password theft has become widespread.

Password theft is a serious problem, frustrating users' expectations of trust, safety, security, and privacy, and interfering with the efforts of ISP's to meet those expectations. While password theft is often involved in particularly serious crimes, such as computer vandalism or identity theft, one of the main motives for password theft is to co-opt user accounts for the sending of 'spam,' *i.e.* unsolicited e-messages that are sent to many recipients at one time, usually to promote a product or service. Most users find spam annoying and intrusive, but it can also be highly offensive, as when one receives spam that advertises web sites that distribute pornography. Furthermore, large volumes of spam severely burden network infrastructures. Thus, control of password theft constitutes an important measure in reducing the volume of spam. A large ISP may have to reset thousands of accounts per week because of stolen passwords, at a cost of several million dollars per year.

Passwords can be compromised in a number of ways. Users often keep their passwords written on easily found POST-IT NOTES (3M Corporation, St. Paul MN) or scraps of paper, instead of committing them to memory; or they may disclose their passwords to friends and co-workers. Often, users are enticed into revealing their

user names and passwords via email, instant messaging or by visiting spurious web sites. Additionally, users may be tricked into downloading and installing Trojan horse programs, *i.e.* keystroke monitoring software that sniffs for typed passwords and subsequently sends them to a hacker. Finally, cracking, guessing a user's password through repeated login attempts, is very common. Generally, cracking is done by means of a software program that iteratively tries one alphanumeric combination after another, until it guesses a user's password.

Crackers generally launch either brute force or dictionary attacks. During a brute force attack, in which the attacker has no previous knowledge of what a password is likely to be, the cracker iteratively tries all possible alphanumeric combinations, starting with one character, and then proceeding to combinations of two characters and so on. A brute force attack typically requires large amounts of computing power. For example, if an attacker were going to attempt cracking all possible six-character passwords composed of the upper-case letters, the lower-case letters and the digits 0-9, there would be about 56,800,000,000 possibilities. Even a powerful computer would require at least several weeks of continuous operation to complete the task. For this reason, brute force attacks are relatively uncommon.

Much more efficient, and more common, are dictionary attacks, in which only words from the dictionary, for example, are tested as passwords. While the total number of words in the English language numbers a few million, there are no more than two hundred thousand words in common use. Thus, a computer capable of trying five thousand passwords per second could try the entire two hundred thousand words in less than a minute. Similarly, the cracker could make guesses based on names, slang, science fiction shows and books, technical jargon and so forth. It is much

easier to take a limited set of words and try combinations based on the known set than it is to crack by brute force.

Most efficient is a rule-based attack, in which the attacker has some knowledge of one or more rules used to generated passwords. For example, a password may consist of a word followed by a one or two digit number, *e.g.* user1, mind67. The cracker then generates passwords according to the rule.

In actuality, most crackers seek to crack the password for a large set of user ID's, for example the passwords of any of the millions of AOL (TIME-WARNER, INCORPORATED) user IDs, rather than try to crack a particular user ID's password. Because many users use one of several simple, common passwords, "abc" or "password," for example, the usual cracking technique is to select such a common password and then iterate through a list of known or generated user IDs, trying each user ID against the common password until a user ID is found that uses the common password. Thus, it is more typical for crackers to guess a user ID matching a chosen password than to guess the password matching a chosen user ID.

For the cracker to target a given service, such as AOL's login servers, the targeted service must offer a higher return, in terms of number of cracked accounts times the value of each cracked account to the cracker, than another service would. This is because a cracker's resources are limited. By reducing the rate at which a cracker is able to crack user accounts, a service can reduce the cracker's return. Any reduction in cracked accounts provides a number of benefits to the service operator, among them reducing the cost of recovering compromised accounts and the cost of handling spam emails sent from them. Additionally, if the service operator can

sufficiently reduce the rate at which accounts are cracked, it should be possible to get the cracker to shift his cracking resources to other, easier targets, and thereby achieve a savings in number of cracked accounts far beyond that achievable through rate limiting alone.

The cracker typically needs to guess possible user ID/password pairs at a very high rate to crack enough accounts to support his spamming or other account-abusing activities. On the other hand, server technology has evolved to help meet threats of this type. Servers and routers have the ability to monitor inbound and outbound traffic flows. A server monitors inbound traffic, for example, by tracking the number of packets per second or bytes per second received, perhaps at a selected port, or from a selected network address. As the number of packets or bits per second in a stream increases, the stream consumes more of the server's bandwidth. To apportion bandwidth appropriately, servers and routers have traffic management capabilities, commonly called rate limiting, which include traffic policing measures. Using rate limiting measures, a server manages inbound and outbound traffic according to preconfigured rate policies that enforce bandwidth limits. Rate policies can be configured for various interfaces such as individual ports or trunk groups. Additionally, rate policies can be configured for various types of traffic, for example a specific MAC (media access control) address, or specific TCP (transmission control protocol), or UDP (user datagram protocol) ports.

At its finest granularity, rate limiting is applied to specific source IP (internet protocol) addresses. In this way, a server can detect traffic coming from a particular IP address that exceeds the bandwidth allocated by the rate policy. Thus, servers recognize atypical traffic and apply rate limiting countermeasures immediately. For

example, a cracker may mount an attack from behind a particular IP address, perhaps behind a proxy server or a firewall, directing requests to the server at a rate that exceeds the allocated bandwidth. The server then limits the rate at which requests are processed from the particular IP address by limiting the bandwidth to that allocated by the rate policy. Additionally, if the traffic exceeds a certain pre-configured rate, the server may drop the traffic entirely. Thus, by imposing rate limiting countermeasures, a server can fend off, or at least, badly frustrate attacks by crackers.

However, a fundamental limitation of IP-address based rate limiting is that a peer IP address is a very weak proxy for a particular user using a particular PC, and hence IP-address based rate limiting is a very blunt and imprecise tool. A counter measure applied to a suspicious IP address can be a draconian measure. Even though each client machine has it's own local IP address, in many cases the internet traffic from these client machines is routed through an intermediary device, such as a firewall or a web proxy server, and it is the IP address of this intermediary device, rather than that of the individual client machine, that is seen as the requesting IP address by the destination network server. Thus, rate limiting traffic from the IP address of the intermediary device effectively denies or limits service to all users whose requests originate from the suspicious IP address, even though the actual target may be a single cracker among them. Thus, by using the peer (requestor's) IP address alone, the network server has no ability to discern which particular client machine made a given request when it is routed through a shared intermediary device. Further, a given user's peer IP address may change from session to session or even from network request to network request. In this case, applying rate limiting to a peer IP address may have no effect on the targeted user.

It would be a great advance in the art to provide a method of fine-grained rate limiting, in which individual user ID/machine combinations could be distinguished and separately rate-limited even when sharing the same peer IP address.

K. Shin, K. Kobayashi, T. Aratani, *Device and method for authenticating access rights to resources*, U.S. Patent No. 5,987,134 (November 16, 1999) provides an approach that requires several different components including challenging data, user identifying information, and an access ticket. Shin, *et al.* are primarily concerned with authenticating a user, rather than establishing a particular piece of hardware as trusted.

M. Ensor, T. Kowalski, A. Primatic, *User-transparent Security method and apparatus for authenticating user terminal access to a network*, U.S. Patent No. 5,721,780 (February 24, 1998) describe a method and apparatus for implementing security in data and telecommunications networks that is independent of and transparent to users. When a user terminal connects to a network control center, the network control center generates an encrypted password based on the user terminal's network coupling identifier. The password is downloaded to the terminal and simultaneously saved by the network control center. Subsequently, when a user attempts to access the network, the network control center retrieves the password from the terminal and compares it with the stored copy. If there is a match, network access is granted. With each logon from the terminal, a new password is generated and downloaded to the user terminal. The exchange of passwords described by Ensor, *et al.* allows a user terminal to be established as trusted on a session-by-session basis. However, the trust is only machine-specific; it is not specific to the

user as well. Furthermore, Ensor, *et al.* fail to contemplate fine-grained rate limiting based on designating a machine as trusted by issuing a trust token.

K. Seamons, W. Winsborough, *Trust negotiation in a client/server data processing network using automatic incremental credential disclosure*, U.S. Patent No. 6,349,338 (February 19, 2002) describe a system in which trust is negotiated between two unfamiliar data processing apparatus by incrementally exchanging credentials. Providing multiple opportunities for exchange of credentials makes it possible to negotiate a higher level of trust between two machines previously unfamiliar with each other than a single exchange of credentials. The approach provided by Seamons, *et al.* involves the iterative exchange of credentials and credential access policies, wherein the credentials are primarily issued by various third parties and describe the holder of the credential.

Siebel 7 Integration with Baltimore SelectAccess 5.0: Technical information brief describes the use of a trust token to provide SSO (single sign-on) functionality across several applications in different domains. Similarly, R. Zuccherato, *Authentication token* (November 6, 2002) describes an authentication token that is issued by an authentication server for later use in authenticating to a different application server. There is no teaching in either source of the use of a trust token to designate a particular machine/user combination as trusted, or differentially applying rate limiting countermeasures based on the presence or absence of a trust token.

It would be highly advantageous to be able to distinguish friendly network traffic from hostile network traffic at the granularity of a single user ID/machine combination. It would be further advantageous to reduce the rate at which legitimate users are

denied service from a server while thwarting attackers operating from behind the same IP address as legitimate users. It would be a still further advantage to improve server defenses and reduce the volume of spam sent and received by network subscribers by implementing a method of fine-grained, trust-based rate limiting.

SUMMARY OF THE INVENTION

The invention provides a method and apparatus for fine-grained, trust-based rate limiting of network requests. Legitimate network traffic is distinguished from untrusted, and potentially hostile, network traffic at the granularity of an individual user/machine combination. Thus, network traffic policing measures such as rate limiting are readily implemented against the hostile or untrusted traffic without compromising service to trusted user/machines legitimately requesting network services ("friendlies").

In one embodiment of the invention, a client successfully authenticating to a server for the first time is issued a trust token by the server. The trust token is typically a cryptographically secure object, such as a certificate, that is both user and machine-specific. Issuance of the trust token, which is stored by the client, establishes the particular user ID/machine pair as trusted. In subsequent logins, the trust token is provided by the client along with the customary authentication information, such as user ID and password. At the server, a rate limiting component applies traffic policing measures, such as adding response latency, according to rate policies that give highest priority to network requests that are designated trusted, based on the availability of a valid trust token. Thus trusted requests are promptly granted. Rate policies may further specify bandwidth restrictions to be imposed for untrusted

network traffic, such as adding a configurable degree of response latency to untrusted traffic. Furthermore, the rate policy can be configured to drop untrusted traffic entirely.

An additional embodiment of the invention is described, wherein trust tokens are stored in a server side database, rather than being stored by the client, allowing the invention to be implemented with clients that lack the capability of storing a trust token. In a further embodiment, a single database provides service to a number of different servers. In a still further embodiment, a single rate limiting component polices traffic for a plurality of servers, while each of the servers manages trust tokens independently of each other.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 provides a schematic diagram of a direct network connection between client and server, without interposition of an intermediary;

Figure 2 provides a schematic diagram of a network connection between client and server wherein the client is behind a firewall or proxy server;

Figure 3 provides a schematic diagram of a process of establishing a client as trusted in a system for trust-based fine-grained rate limiting of network requests according to the invention;

Figure 4 provides a schematic diagram of a process of updating a trust token in the system of Figure 2 according to the invention;

Figure 5 shows provides a diagram of an exemplary trust token from the system of Figure 2 according to the invention;

Figure 6 provides a block diagram of a server architecture from the system of Figure 3 according to the invention;

Figure 7 provides a flow diagram of an authentication process according to the invention;

Figure 8 provides a schematic diagram of a process of establishing a client as trusted, wherein an issued trust token is stored in a server-side database according to the invention;

Figure 9 depicts an enhanced rate limiting approach that may be applied to untrusted requests;

Figure 10 provides a schematic diagram of a centralized system for trust-based fine-grained rate limiting of network requests according to the invention; and

Figure 11 provides a schematic diagram of a system trust-based fine-grained rate limiting of network requests wherein untrusted logins are routed to a central rate limiting server according to the invention.

DETAILED DESCRIPTION OF THE INVENTION

As shown in Figures 1 and 2, a user of a client machine 101 requests network access or services from a server 102. It will be appreciated that each client 101 has a distinct IP address, denoted by IP_1 - IP_n. When a client directs a request to a server directly, without the interposition of any intermediary devices, as in Figure 1, the server 102 is readily able to discern the source IP address. Often, however, the client 101 sends its request to the server 102 from behind a proxy server or firewall 103, wherein the proxy/firewall 103 has a distinct IP address, for example "1.160.10.240." There may be many users and many machines behind the proxy/firewall 103, all directing requests to the server 102. The fact that all of this traffic originates from behind the proxy firewall 103 has the effect that all of the traffic, even though it originates from many different users and many different machines, is seen by the server 102 to originate from the same source IP address, that of the proxy/firewall 103, *i.e.* 1.169.10.240. The server cannot distinguish traffic originating from one machine behind the proxy/firewall 103 from traffic originating from another machine; it all looks the same to the server.

One assumes that the majority of users behind the proxy/firewall 103 are "friendlies," users having legitimate user accounts on the server 102 and having benign intentions. It is also possible, however, that hostile traffic can originate from behind the proxy/firewall 103, for example an attacker trying to crack passwords to user accounts on the server 102. As previously described, an attack by a cracker can consume enough of the server's bandwidth so that the attack is readily detected, and the server can implement countermeasures to defend against the attack, such as rate limiting.

Rate limiting is a traffic management tool that allows a server or router to control the amount of bandwidth specific traffic uses on specific interfaces. While the specific implementation of rate limiting varies according to the manufacturer of the server or router, some general principles of rate limiting are described herein below.

Many network devices implement rate limiting using a token bucket model, in which arriving packets are admitted to a port by removing admission tokens from a bucket. It will be appreciated that larger packets or a greater number of packets per second require more tokens. Tokens are added to the bucket at a rate set by the configured rate limit. As long as the bucket is full, no additional tokens can be added. Traffic arriving at the port can only be accepted if there are sufficient tokens in the bucket. Thus, a rapid burst of traffic causes rapid depletion of the bucket. If the bucket is emptied, traffic cannot be granted ingress to the server. In some cases, incoming traffic is simply dropped until the bucket is replenished. In other cases, incoming traffic is queued until the bucket contains sufficient tokens to admit a portion of the queued traffic. It will be appreciated that queuing incoming traffic and replenishing the bucket with tokens at a configured rate has the effect of imposing latency on the rate at which the incoming traffic is processed. Thus, the latency is configurable by configuring the rate at which the bucket is replenished.

The above scheme for implementing rate limiting is provided for the sake of description only. Other methods of rate limiting are commonly known by those having an ordinary level of skill in the art and are consistent with the spirit and scope of the invention.

Network devices such as servers generally have capabilities for both fixed and adaptive rate limiting. Fixed rate limiting enforces a strict bandwidth limit. The device forwards traffic that remains within the limit, but either queues or drops all traffic that exceeds the limit. Adaptive rate limiting enforces a flexible bandwidth limit that allows for bursts above the limit.

FIXED RATE LIMITING

Fixed rate limiting allows one to specify the maximum number of bytes a given port on a network device, such as a server, can send or receive. The port drops or queues bytes that exceed the amount specified. Additionally, rate limiting can be configured for both inbound and outbound traffic. A rate limit is usually expressed in bits per second. The fixed rate limiting policy generally applies to one-second intervals and allows the port to send or receive the number of bytes specified by the rate policy.

Fixed rate limiting counts the number of bytes that a port either sends or receives within the configured time interval. If the number of bytes exceeds the maximum number specified, the port drops all further packets for the rate-limited direction for the duration of the specified interval. At the end of the interval, the port clears the counter and re-enables traffic.

Generally, rate policies are configured on a port-by-port basis. Each port is configured by means of an editable configuration file. Rate policies are typically configured by entering scripted commands in the port's configuration file that specify the direction of traffic flow and the maximum number of bits per second.

ADAPTIVE RATE LIMITING

Adaptive rate limiting rate allows configuration of the amount of traffic of a given type a specific port can send or receive. It also allows changing the IP precedence of traffic before forwarding it. Rate policies can be configured for specific types of traffic, for example: layer 3 IP traffic, specific source or destination IP addresses or networks, specific source or destination TDP or UDP ports, specific MAC (media access control) addresses, specific IP precedence values, or Diffserv (differentiated service) control points.

Configuration of adaptive rate limiting policies is similar to that for fixed rate limiting, except that rate policies are specified for each type of traffic. In the event that a rate policy is not specified for a particular traffic type, that type of traffic is sent and received without rate limiting.

Rate limiting should ideally be applied at as fine a granularity as possible, so that those actually doing the cracking are strongly limited without limiting, *i.e.* inflicting "collateral damage" upon, friendlies in the virtual neighborhood. Unfortunately, this is the biggest limitation of IP address-based rate limiting schemes, because IP addresses are not fine-grained when it comes to network login requests, as previously described.

In practice, this means that it is necessary to identify proxy/firewall IP addresses. This is accomplished by monitoring logs and user feedback for selected network services, and attempting to tune their rate limits such that they are loose enough to account for an average number of users behind such a proxy IP address, yet tight enough to still have some effect on crackers from behind those IP addresses. This is

a challenging, ongoing, error-prone exercise that results in too many friendlies being denied service and too many crackers not being adequately rate limited. This scheme also provides an opportunity for malicious hackers to effect a denial of service (DOS) attack on users behind the same proxy IP address.

DIFFERENTIATING BAD FROM GOOD TRAFFIC

IP-address-based rate limiting entails identifying the "bad" IP addresses and then enacting rate limiting countermeasures upon them. Crackers know about this and thus use various means, for example, rotating among proxies or using many different IP addresses, to avoid sending network requests from "bad" IP addresses. Fundamentally, identifying and tagging the "bad" is much more challenging than identifying and tagging the "good," because the bad can and do apply their resourcefulness to escape the tagging. Conversely, in an alternative approach where only the "known good" users are tagged in a cryptographically secure way and are rewarded for being good, there is nothing that the bad users can do, because there is no bad tag to avoid.

TRANSITORINESS

IP addresses typically cannot adequately represent the entities to which it would be ideal to apply trust-based rate limiting to: specific users on specific computers. Many users are assigned different IP addresses each time they start new sessions, for example dialup users; or each time they restart their computers, for example consumer broadband users. Other users randomly come from 1 of *n* IP addresses corresponding to the machines in the proxy or firewall farm through which their requests originate. Thus, IP address-based countermeasures are only good for as

long as the IP/user binding persists. After that, an attacker gets a fresh start and suffers no penalty for past transgressions.

A NEW APPROACH

The invention recognizes the following:

Cracker login attempts are distinguishable from typical friendly login attempts:

- All cracker login attempts originate from a computer from which a to-be-cracked user ID has not previously logged in successfully;
- Conversely, most friendly login attempts come from a computer from which the user ID has previously logged in successfully.

Incremental login response latency is very harmful to crackers, but minimally harmful to friendlies:

- Crackers' success rates are directly proportional to number of guesses they can make in a given unit of time;
- Crackers need to make a lot of rapid guesses in order to cost-effectively crack accounts;
- Slowing down cracker guessing proportionally reduces their success rate; and
- At some point, it is more cost-effective for crackers to move to easier targets.

As shown in Figure 3, the inventive approach is to tag each user ID/ computer pair that successfully logs in, such that future login attempts by that user ID on that computer are treated as trusted and thus incur no additional login response latency, and to add a configurable amount of incremental latency to all untrusted, *i.e.* untagged, login attempts. In other words, cracker logins are slowed down without

affecting most friendly logins from trusted computers, and only slightly penalizing the relatively few untrusted friendly logins.

Preferably, incremental latency is added to both successful and unsuccessful login attempts from untrusted clients. If it is only added to unsuccessful ones, the crackers quickly realize that if a response is not received within a successful response time, the response can be assumed to be unsuccessful. Thus, the cracker need not continue waiting the extra time for the actual response. In one embodiment of the invention, latency is added only to a configurable percentage of successful logins. Thus, attempting not to miss an opportunity for a correct guess, a cracker is forced to wait for responses for wrong guesses, while latency is attached only to some percentage of friendlies that login successfully. In an alternate embodiment, latency is added only to requests from IP addresses that have exceed a configurable, "add-latency" login rate, as described below.

Referring now to Figure 3, a process is shown for:

- identifying entities legitimately entitled to service; and
- establishing those identified entities as trusted.

By way of a client 101, a sever requests access or services from a network device, such as a server 102. The request takes the form of a login request 103. In an exemplary embodiment, the login request includes a user ID, a password, and a client ID. A client may include a desktop or notebook computer, a handheld device such as a palmtop computer or a personal organizer, or a wireless telephone. For purposes of the invention, a client is taken to mean any device or software application through which a user requests services from the network device 102. A

network device, in addition to being a server, could also include a router or a switch. The client ID constitutes one or more items of data that can be used to uniquely or semi-uniquely identify the client. Thus, a user ID/client pair represents a unique or nearly unique entity. The request is received at the network device 1-2 at time T_1. Successfully logging in to the network device 102 identifies the particular user ID/client pair as being entitled legitimate access to the network device 102. Instead of the client providing a password as part of the login credentials, a derivative of the password may be supplied, such as a cryptographic hash of the password.

Following the successful login, the network device issues a user-specific, machine-specific trust token 104. At time T_2, the trust token 104 is stored on the client, thus establishing the particular user ID / client pair as trusted.

Referring now to Figure 4, a schematic of a trusted login is shown. As in Figure 3, the user requests service from the network device 102. The login request 103, however, is accompanied by the previously stored trust token 104. By evaluating the trust token 104, the network device is signaled that the login request originates from a trusted entity, a user ID/client pair legitimately entitled to service from the network device 102. Following successful login, an updated trust token 105 is transmitted to and stored on the client 101.

Referring now to Figure 5, an exemplary embodiment of a trust token 500 is shown. The exemplary trust token constitutes an object, such as a certificate or a statement. As shown, the data included in the trust token includes:

- user ID;
- timestamp of first login;

- client ID; and
- timestamp of most recent login.

The foregoing description of the trust token is intended only to be descriptive. Other types of data could be included in the trust token and are well within the spirit and scope of the invention.

Figure 6 provides a block diagram of an exemplary functional architecture for a network device 102. In an exemplary embodiment, the network device includes:

a trust assessor 601;

a rate limiter 602;

an authenticator 603; and

a trust grantor 604.

Figure 7 illustrates a typical request processing flow as performed by these components, but it should be appreciated that many alternative processing flows are possible and are consistent with the spirit and scope of the invention. For example, Figure 7 shows a process flow involving a client that is capable of storing and transmitting a trust token. An embodiment of the invention is described further below wherein the trust token is stored in a server-side database, indexed according to client ID and user ID. In such case, the trust assessor 601 retrieves the corresponding trust token from the database, based on the user ID and client ID stored in the database. The token grantor 604 stores an issued token in the database, indexing it according to client ID and user ID.

The trust assessor 601 attempts to obtain the requesting entity's trust token, if any, from either the request or from the server's database (if stored server-side). If it finds a trust token, it proceeds to validate the token's cryptographic integrity and authenticity; for example, by attempting to decrypt it with the secret key with which it was encrypted. If cryptographic validation succeeds, the trust assessor 604 then proceeds to perform other validation checks defined by the server, such as checking that the client ID contained in the token matches that provided on the request, and checking that the token's first-used and last-used timestamps are no earlier than the server's configured minimum timestamps.

The rate limiter 602 prioritizes authentication requests to be processed by the authenticator 603, based on the client ID, the presence and assessed validity of the associated trust token, and any defined rate limiting policies, to determine if and how to rate-limit the requests. A request accompanied by a valid token is passed immediately to the authenticator 603, with little or no rate limiting being imposed. A request lacking a valid token is given a lower priority. As denoted by the arrow 703, the rate limiter may completely reject a request, as determined by the rate policies configured for the server.

The authenticator 603 is responsible for verifying authenticity of the login credentials, typically user ID and password, presented by the client. The authentication element 503 consults either an internal or external data store containing the authorized user ID and password and determines whether the credentials supplied match those stored in the data store. If the login credentials are invalid, the request is rejected, as denoted by the arrow 702

The trust grantor 604 generates a fresh trust token for the authenticated entity. The trust grantor 604 initially collects the contents of the token supplied by the authenticated entity during authentication, including any of user ID, client ID, first-used timestamp from the old trust token and the current time, *i.e.* the last-used time to put into the token, and then optionally encrypts the contents using a cryptographically secure encryption algorithm and encryption key. Preferably, the generated trust token includes both the above mentioned encrypted data as well as unencrypted plain-text that denotes at least the identifier of the encryption key used. Additionally, the token issuer generates trust tokens for successful, but previously untrusted logins. In either case, the generated trust token is either transmitted to the client, as denoted by the arrows 701 and 704, or stored in the server-side database, indexed as previously described.

While Figure 6 depicts the server-side as including a single unit having a number of functional components, in actuality the server, or network device could also include a number of different units, in which the various server side functions are distributed across units. Examples of distributed server-side architecture are described further below.

There follow herein below descriptions of a number of embodiments of the invention involving different server architectures and different client types.

TOKEN-CAPABLE CLIENT

A token-capable client, as shown in Figures 3 and 4, is any client capable of storing and transmitting a cryptographically secure trust token, such as a web browser. In a preferred embodiment of the invention, after a client logs into the login server

successfully, the login server generates a cryptographically secure trust token, for example triple DES encrypted (data encryption standard). The trust token, shown in Figure 5, typically contains at least the user ID and the timestamps of first and last logins by the user ID on the client. Additionally the token contains some type of unique, non-spoofable client ID, such as the MAC (media access control) address or the disk drive serial number, such that replay of the token from a machine other than the client the token was issued to is discernable to the server. However, as long as the tokens are sufficiently difficult to capture, as is the case when the transmission occurs over an encrypted communication channel, it is not absolutely necessary that the data be non-spoofable.

Figure 8 illustrates another embodiment of the invention, wherein the server stores the trust tokens in a server-side repository 801, such as a database, instead of sending them down to the client for storage on the client. In this case the server indexes stored tokens by the associated entity ID, preferably the user ID + client ID. Further, the client ID may be client-specific data originated by the client, *e.g.* the MAC address of the client computer, or may be a unique client ID 802 assigned by the server and sent down to the client for storage and return to the server on subsequent authentication requests.

ANONYMOUS CLIENT

In the case of logins from clients that provide no uniquely or semi-uniquely identifying client data, and which are not capable of storing and replaying server-generated trust tokens, the following IP-based trust scheme is provided:

Following each successful authentication, the network service adds or updates a database record containing the following information:

- user ID;
- client IP address;
- timestamp of first successful login by user from this client IP address; and
- timestamp of last successful login by user from this client IP address.

A record is added if it's the first time user has successfully authenticated from the client IP address. Otherwise, the record having matching user ID and client IP address is updated.

Each new authentication request contains the user ID and the client IP address. The network service extends trust to a new request according to one of the following alternate schemes:

- Exact client IP match. If the user ID and client IP address of the request match exactly those of an existing database record, and if the timestamps from the database record pass the configurable timestamp bounds checks, the request is trusted; otherwise, it is not.
- Trusted IP address range match:
    - Step 1: determine trusted IP addresses ranges for the user ID. Network service computes one or more trusted IP address ranges for the user ID from the set of successful authentication records stored for that user ID.
    - Step 2: check client IP address. If the client IP address of the current request is in one of the trusted IP ranges, the request is trusted, subject to step 3, otherwise it is not.

- Step 3: check timestamps. Compute the earliest authentication timestamp for the matching trusted IP address range as the minimum of the earliest authentication timestamps of the matching records, and likewise for the latest authentication timestamp. If the timestamps pass the configurable timestamp bounds checks, the request is trusted, otherwise, it is not.

The process flow for establishing an anonymous client as trusted is analogous to that shown in Figure 7. Instead of attempting to obtain a trust token for the entity attempting to authenticate, the trust assessor 601 checks the database for a record of a previous successful authentication from the entity. The remaining steps proceed as previously described; however the role of the trust grantor 604 is creation or updating of a database record, rather than issuance of a trust token.

Figure 9 depicts an enhanced rate limiting approach that may be applied to untrusted requests. The arrow 901 is included to indicate an increasing volume of network requests.

Below a configurable login threshold 'A,' no latency is applied. Traffic below the first threshold is assumed to be trusted, based on the low request rate per unit of time. The assumption tends to be valid because user ID/client pairs legitimately entitled to request service would successfully be able to login on the first attempt. Thus, the traffic volume tends to be low.

Above a second configurable threshold 'B,' traffic is dropped altogether, on the assumption that the traffic is hostile. The assumption tends to be valid because the

rate specified by threshold 'B' is one achievable only by automatically generated network traffic, such as would result during an attack by a cracker or during a denial of service attack.

Above threshold 'A' and below threshold 'B,' latency is applied to untrusted login requests. Network traffic within this range is not of a high-enough volume to justify the extreme measure of completely blocking it, but is high enough to justify in increased index of suspicion. As previously described, imposing login latency frustrates a cracker's efforts sufficiently to give them a strong incentive to seek out weaker targets.

IDENTIFIABLE CLIENT

While the previous embodiments of the invention are applicable to the majority of clients, a third embodiment shown in Figure 8, is applicable to clients that are not token-capable, but that do provide uniquely or semi-uniquely identifying data about the client making the login request. Examples of uniquely identifying data include:

- Disk drive serial number;
- MAC address.

In the case of semi-uniquely identifying data, different items of data can be concatenated to form an identifier. For example, the client build number, the PC RAM (random access memory) and the phone number dialed are concatenated to produce a unique or semi-unique identifier. As shown, the client 101 directs a login request 103 to the network device 102. As previously described, the network device issues a trust token. However, as indicated by the arrow 802, the trust token is stored in a server side database. In this case, the login server uses the user id plus

client data as a key into a database 601 of trust tokens stored server-side, instead of the token being stored on the client. During subsequent logins, as indicated by the arrow 803, the trust token is retrieved from the database 601 and evaluated. Following validation, the trust token is updated as previously indicated, and access is granted to the client 101. As shown in Figure 10, an embodiment of the invention is possible in which a single centralized database 1001 serves a number of network devices, denoted here by servers 1002a – 1002c.

While the incremental latency penalty is the currently preferred embodiment of the invention due to its ease of implementation and its user-friendliness, it should also be appreciated that it is possible to impose other penalties upon an untrusted login attempt. One such penalty is a requirement that the client successfully complete a Turing test. A Turing test typically constitutes a question or a task that can be readily solved by a live human, but that could be solved only imperfectly, if at all, by a machine. Thus, by requiring successful completion of a Turing test prior to login, one is readily able to distinguish a human user that is more likely to be entitled access to the server from a cracker launching an attack using cracking software. For example, a Turing test might be to correctly recognize characters embedded in an image containing distorted text characters to obtain permission to attempt a login.

ENHANCEMENTS AND EXTENSIONS

Figure 11 shows an alternate embodiment of the invention in which all business logic and code involved with processing untrusted logins is centralized to a rate limiting server 1101. Thus, the servers 1102a – 1102c handle trusted logins, and all untrusted logins are directed to the central server 1101.

It will be appreciated that transmission of sensitive data such as authentication credentials most appropriately occurs over a secure network connection. Thus, all embodiments of the invention rely on protocol that ensures security and privacy in network communications. The invention preferably employs the SSL (secure sockets layer) protocol.

Although the invention has been described herein with reference to certain preferred embodiments, one skilled in the art will readily appreciate that other applications may be substituted for those set forth herein without departing from the spirit and scope of the present invention. Accordingly, the invention should only be limited by the Claims included below.